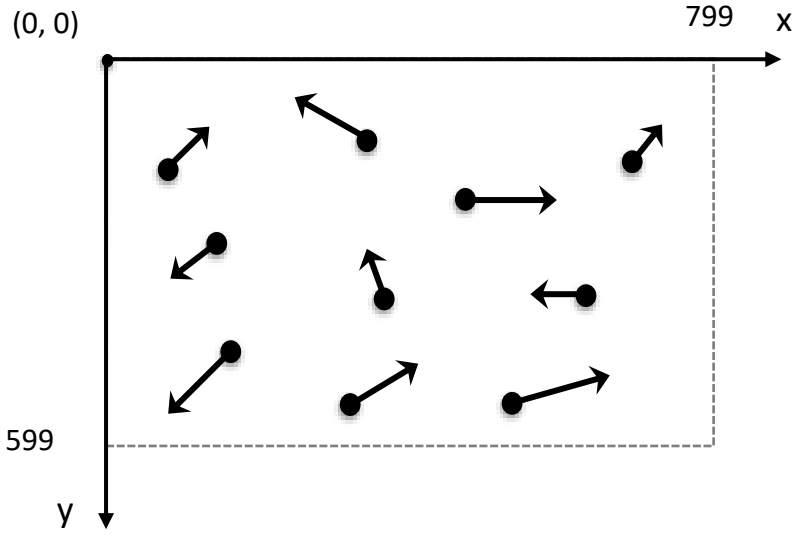
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMAÇÃO DE COMPUTADORES

TRABALHO PRÁTICO 2

Enquanto estudava Geometria e Álgebra, Joãozinho se deu conta de que os vetores podem ser usados para guiar o deslocamento de objetos em um plano, e viu que a tela do computador pode ser vista como um plano em que o eixo y cresce para baixo. Em programação de computadores, Joãozinho aprendeu que o computador é uma ferramenta capaz de resolver problemas que exigem muitas repetições de uma determinada tarefa. A partir daí, ele teve a ideia de construir uma simulação. Ele quer simular o deslocamento de várias partículas dentro de um espaço confinado. Ajude-o a construir esta simulação seguindo as instruções abaixo.



Dez partículas devem ser criadas dentro de um espaço de tamanho 800 x 600. O nome das partículas, suas posições iniciais, o sistema de coordenadas utilizado, os valores do seu vetor deslocamento e o código da sua cor devem ser lidos pelo teclado. As particulas devem ser **armazenadas em um vetor estático** de tamanho 10**.**

Exemplo de entrada:

```
Alan Turing:       790  101 P  40.0  20.0  0
Ada Lovelace:      480   76 P 138.0  25.0  1
Donald Knuth:      210  563 C  19.0  23.2  2
Peter Naur:        451  421 P 284.0  30.0  3
Ken Thompson:      630   46 C -22.1  14.3  4
Von Neumann:       100  500 P  89.0  17.9  0
Dennis Ritchie:     58  325 C  28.0 -25.0  1
Bjarne Stroustrup: 128  211 C -24.5 -25.9  2
Brian Kernighan:     2  182 C  35.5   2.6  3
Leslie Lamport:    737  555 P 149.0  18.2  4
```

Os dois pontos após os nomes são obrigatórios. Os nomes são sempre compostos.

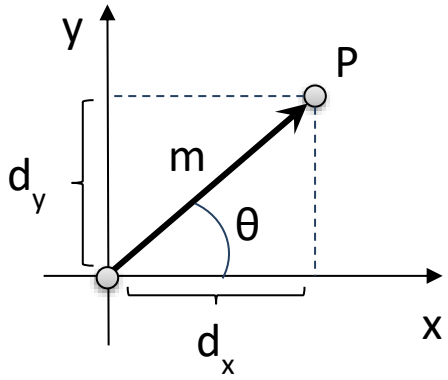
Os espaços em branco extras, para alinhar os números, não precisam ser digitados, estão aí apenas para deixar a entrada mais clara.

Para trabalhar com uma partícula, construa uma biblioteca (particula.h e particula.cpp):

- Represente uma partícula por um registro que guarde as seguintes informações:
  - Nome (vetor de 20 caracteres)
  - Posição atual no eixo x e y (registro com pontos-flutuantes)
  - Sistema de coordenadas utilizado (caractere P ou C)
  - Vetor deslocamento (união de registros: polar ou cartesiano)
  - Cor da partícula (enumeração com 5 cores)
- Construa uma função **operator>>** para ler valores do tipo partícula

O vetor deslocamento pode ser representado por um ângulo e uma magnitude (coordenadas polares) ou por um ponto no plano (coordenadas cartesianas). Se quisermos deslocar uma partícula da origem dos eixos até uma posição P, basta somar a posição atual (x, y) da partícula com os valores d_x e d_y, sendo estes dados por:

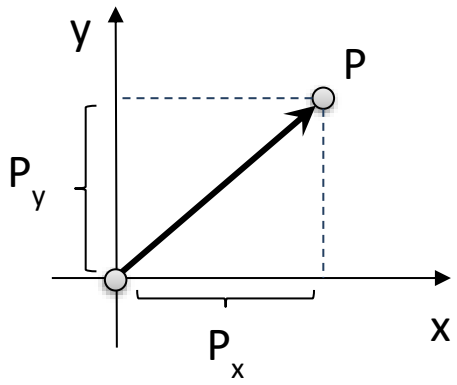
Coordenadas Polares



$$d_x = m * \cos\theta$$

$$d_y = m * \text{sen}\theta$$

Coordenadas Cartesianas



$$d_x = P_x$$

$$d_y = P_y$$

Para trabalhar com o vetor deslocamento, construa uma biblioteca (vetor.h e vetor.cpp):

- Use um **registro** para representar um vetor em coordenadas **polares**
- Use um **registro** para representar um vetor em coordenadas **cartesianas**
- Use uma **união** para **representar um vetor** em uma das coordenadas
- Use um **registro** para representar as coordenadas (x, y) de um **ponto na tela**
- Construa uma **função** que receba um **ponto na tela**, um **vetor deslocamento** e o **tipo das coordenadas** utilizadas pelo vetor. A função deve utilizar uma das funções abaixo para obter e retornar o deslocamento do ponto pelo vetor:

    - Construa uma **função** que receba um **ponto na tela**, um **vetor deslocamento** em coordenadas **cartesianas** e retorne o resultado do deslocamento do ponto pelo vetor.
    - Construa uma **função** que receba um **ponto na tela**, um **vetor deslocamento** em coordenadas **polares** e retorne o resultado do deslocamento do ponto pelo vetor.
- Construa uma função **operator>>** para ler o tipo usado para coordenadas cartesianas
- Construa uma função **operator>>** para ler o tipo usado para coordenadas polares

Para trabalhar com as cores das partículas, construa uma biblioteca (cor.h e cor.cpp)

- Crie um tipo enumeração com cinco constantes para cores de sua escolha
- Construa uma função **operator>>** para ler o tipo usado para as cores

A cada passo da simulação, um número aleatório entre 1 e 10 deve ser sorteado. Esse número é a quantidade de particulas que serão movidas nesse passo da simulação.

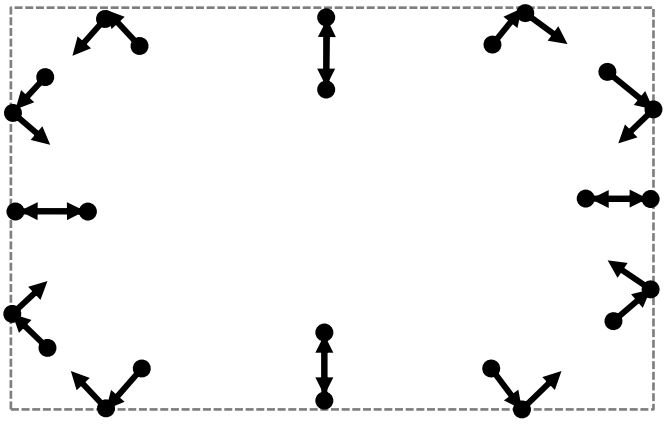
- O número deve ser usado para criar um vetor dinâmico de ponteiros para partículas. Depois deve-se sortear que partículas serão movidas. Os endereços das partículas sorteadas devem ser guardados nesse vetor dinâmico.
- Utilize um laço para percorrer o vetor dinâmico, acessar e movimentar as partículas apontadas. Cada partícula deve ser movida pelo valor das componentes do seu vetor deslocamento, utilizando as funções implementadas na biblioteca Vetor.

O programa deve exibir o número do passo da simulação, quantas partículas foram sorteadas, quais posições foram sorteadas, quantas partículas tocaram nas bordas (ou sairam) da área de confinamento e os respectivos nomes dessas partículas, como mostrado no exemplo abaixo:

```
Simulação
---------
#1: 5 ( 0 2 7 4 8 ) = 0
#2: 2 ( 9 2 ) = 0
#3: 4 ( 0 8 3 7 ) = 1 ( Knuth )
#4: 8 ( 2 4 1 0 3 5 9 7 ) = 2 ( Naur Turing )
...
```

Use uma linha para cada passo da simulação. Exiba jogo da velha, dois pontos, parênteses, sinal de igual e espaços, exatamente como no exemplo.

Quando uma das partículas tocar nas bordas ou sair do espaço de confinamento, a sua direção deve ser modificada de forma que ela siga na direção oposta ao lado em que colidiu:



Para atingir esse resultado, construa **funções** na biblioteca vetor para inverter a direção de um vetor no eixo horizontal e no eixo vertical, usando as coordenadas cartesianas e polares:

- **Coordenadas cartesianas**: inverter o vetor no eixo horizontal ou vertical significa multiplicar a coordenada x (horizontal) ou y (vertital) do vetor por -1.

- **Coordenadas polares**: primeiro obtenha as coordenadas cartesianas equivalentes (d_x e d_y), faça a inversão do sinal como descrito acima, e converta de volta para coordenadas polares. Como a magnitude do vetor não muda, a conversão em coordenadas polares precisa calcular apenas o ângulo através da fórmula ângulo = $\tan^{-1}(dy/dx)$.

O teste da colisão da partícula com as bordas do espaço de confinamento pode ser feito com as instruções abaixo:

```
// partícula colidiu no eixo horizontal
if (x >= 800 || x <= 0)
{
}

// partícula colidiu no eixo vertical
if (y >= 600 || y <= 0)
{
}
```

A simulação deve executar até que 100 passos tenham resultado em alguma colisão. O resultado de cada passo que teve colisão deve ser **armazenado em um vetor estático** de 100 elementos. O resultado deve ser composto pela quantidade de partículas que colidiram, um vetor estático de 10 ponteiros para partículas, e o número de passos transcorridos na simulação.

Para mostrar o nome das partículas que colidiram, construa uma **função** que receba um vetor de caracteres e mostre na tela apenas o último nome. Assuma que o vetor receberá sempre nomes compostos por nome e sobrenome.

Ao final da simulação, exiba as seguintes estatísticas:

- O número total de colisões da simulação
- O número médio de partículas que colidiram a cada passo
- A quantidade média de passos entre uma colisão e outra
- O número de colisões para cada cor de partícula

Os resultados do programa devem seguir o formato abaixo:

```
Resultados
----------
Colisões: 114
Colisões por passo: 0.264501
Média de passos entre as colisões: 4.34343
Colisões por cor: Azul(16) Verde(29) Preto(27) Roxo(23) Branco(19)
```

EXIGÊNCIAS

- Não use variáveis globais
- Não escreva mensagens solicitando a entrada de dados
- A solução do problema deve utilizar as funções que foram pedidas
- Use comentários, indentação e deixe o código organizado
- Libere toda a memória alocada dinamicamente
- Não utilize #pragma once nos arquivos de cabeçalho
- Utilize as diretivas #ifndef, #define e #endif para proteger os arquivos de cabeçalho contra múltiplas inclusões

ENTREGA DO TRABALHO

Grupos: Trabalho individual

Data da entrega: 03/12/2019 (até a meia-noite)

Valor do Trabalho: 3,0 pontos (na 2a Unidade)

Forma de entrega: enviar apenas os arquivos fonte (.cpp) e os arquivos de inclusão (.h) compactados no formato **zip** através da tarefa correspondente no SIGAA.

O não cumprimento das orientações resultará em **penalidades:**

- Programa não executa no Visual Studio 2019 (3,0 pontos)
- Programa contém partes de outros trabalhos (3,0 pontos)
- Atraso na entrega (1,5 pontos por dia de atraso)
- Arquivo compactado em outro formato que não zip (0,5 ponto)
- Envio de outros arquivos que não sejam os .cpp e .h (0,5 ponto)
- Programa sem comentários e/ou desorganizado (0,5 ponto)